Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aljaž Fajmut

Razvoj partnerskega sistema na temelju modernih razvojnih metodologij

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Peter Peer

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Razvoj partnerskega sistema na temelju modernih razvojnih metodologij

Tematika naloge:

Partnerstvo med različnimi prodajalci in uporabniki je zelo razširjena in popularna metoda trženja. Uporabniki veliko lažje in hitreje kupijo nek izdelek, če jim izdelek priporoča nekdo, ki mu zaupajo. Sistem s to funkcionalnostjo imenujemo partnerski sistem. V diplomi implementirajte spletni partnerski sistem, ki zaslugo za obisk strani ali nakup izdelka pripiše ustreznemu promotorju ter vodi evidenco provizij. Naredite pregled obstoječih sistemov ter zasnujte svojega na podlagi modernih razvojnih metodologij, pri tem pa sledite cilju, da bo razvit sistem enostaven za integracijo v obstoječe sorodne spletne storitve.

Rad bi se zahvalil svojima staršema - Irmi in Stanku, ki sta mi nudila podporo in razumevanje na poti šolanja in odraščanja. Zahvalil bi se tudi izr. prof. dr. Petru Peeru za odprtost, razumevanje, strokovnost ter podporo pri izobraževanju in nastajanju diplomskega dela. Želim si, da bi takih profesorjev bilo več.

Diplomo posvečam svoji mami Irmi.

Kazalo

Povzetek

Naslov:
Razvoj partnerskega sistema na temelju modernih razvojnih metodologij

Partnersko sodelovanje je priljubljena in učinkovita vrsta trženja izdelka ali storitve prek poslovnih partnerjev. Diplomsko delo opisuje razvoj produkta, ki nam omogoča, da lahko v obstoječo spletno aplikacijo (bodisi trgovino ali storitev) enostavno implementiramo funkcionalnost partnerskega sistema, kar nam omogoči vrsto novih možnosti pri trženju naših storitev ali izdelkov. Sistem je zasnovan tako, da zahteva minimalno količino posegov za integracijo v obstoječo aplikacijo, razvoj produkta pa je potekal na temelju modernih razvojih metodologij, ki smo jih uporabili tako, da smo na čimbolj učinkovit način prišli do minimalnega fukcionalnega produkta v čimkrajšem času.

Ključne besede: partnerski sistem, zagonsko podjetje, podjetništvo, trženje, moderne razvojne metodologije.

Abstract

Title: Development of the affiliate system based on modern development methodologies

Affiliate partnership is a popular and effective method of online marketing through affiliate partners. The thesis describes the development of a product, which allows us to easily integrate affiliate system into an existing platform (e-commerce or service). This kind of functionality opens up growth opportunities for the business. The system is designed in a way that it requires minimal amount of changes for the implementation into an existing application. The development of the product is based on the modern development methodologies, which are used so that we get the minimum viable product in the most efficient and quickest way.

Keywords: affiliate system, startup, business, marketing, modern development methodologies.

Poglavje 1

Uvod

Partnerski sistem (angl. affiliate system) je funkcionalnost, ki omogoča, da določen izdelek ali storitev lahko promovirajo njegovi uporabniki ali prodajalci, ki so za to specializirani in ob vsaki prodaji tega izdelka zaslužijo provizijo.

Partnerstvo je med različnimi prodajalci in uporabniki [1] (angl. affiliate marketing) zelo razširjena in popularna metoda trženja. Uporabniki, če jim izdelek priporoča nekdo, ki mu zaupajo, tudi veliko lažje in hitreje kupijo. Tovrstno partnerstvo se na spletu ponavadi pojavlja v obliki križanih promocij [2] (angl. cross-promotion). Povežeta se dve podjetji, ki imata podoben segment uporabnikov in vsak svoji uporabniški bazi priporoča izdelek svojega partnerja. Tako lahko obe enostavno razširita krog svojih uporabnikov in hkrati zaslužita provizijo s promocijo partnerskega izdelka. Takšne akcije so običajno veliko bolj učinkovite kot pa kakršnokoli oglaševanje na spletu.

Poleg tega načina je na spletu tudi ogromno posameznikov, ki so specializirani in služijo izključno samo od partnerskih promocij (angl. affiliates). Takšni promotorji so ponavadi pisalci blogov (predvsem ocen in primerjav izdelkov), lastniki raznih nišnih forumov ali portalov.

Na spletu se partnerski sistemi vežejo na spletne strani in delujejo tako, da po registraciji uporabnik (promotor) dobi posebno povezavo na spletno stran, kjer se izdelek prodaja. Povezava vključuje identifikacijski atribut,

ki zaslugo za obisk strani ali nakup izdelka pripiše določenemu promotorju. Tak partnerski sistem omogoča promotorjem pregled nad statistiko obiskov in prodaj uporabnikov, ki so jih napotili k določenemu izdelku ali storitvi.

Obstaja veliko različnih partnerskih sistemov, ki se v glavnem ločijo na samo-gostovane [3] (angl. self-hosted) in upravljano-gostovane sisteme [4] (angl. managed-hosted). Večina prvih je spisanih v PHP jeziku in so večinoma zastareli, zato je tudi njihova prilagoditev zelo otežena. Problem z upravljano-gostovanimi sistemi pa je, da ponudniki zaračunavajo visoke zneske za uporabo, poleg tega pa v večini poberejo tudi svojo provizijo od prodaj. Druga velika pomanljivost obeh sistemov je, da zahtevajo radikalne spremembe v funkcionalnosti nakupnega procesa spletne strani za implementacijo partnerskega sistema.

V podjetju se s partnerskimi programi ukvarjamo od leta 2007, resno pa smo se začeli ukvarjati z razvojem SaaS (Software as a Service) orodij leta 2011. V tem času smo opazili omenjene pomankljivosti tovrstnih sistemov in spoznal načine, kako bi bilo mogoče nekatere med njimi učinkovito odpraviti.

Zaznali smo tudi potrebo po takšnemu sistemu v povezavi z našimi obstoječimi rešitvami. Prejeli smo veliko elektronskih sporočil od uporabnikov, ki jim je naša storitev všeč. Sprašujejo nas o takšnem sistemu in iščejo ustrezno rešitev. Ideja, kako to težavo odpraviti (za nas in tudi za druge) na veliko bolj učinkovit način, kot pa je to mogoče z obstoječimi rešitvami, je predstavljena in ovrednotena v tem delu.

Prišli smo tudi do spoznanja, da lahko z analitičnim orodjem Mixpanel [5] poenostavimo implementacijo partnerskega sistema z minimalnimi spremembami v kodi. Poleg tega nam ta integracija omogoča še mnoge prednosti, ki bodo razložene v nadaljevanju diplomske naloge.

1.1 Partnerski sistemi

V grobem lahko partnerske sisteme razdelimo na samo-gostovane [3] (angl. self-hosted) in upravljano-gostovane [4] (angl. managed-hosted). Vsaka vrsta

ima določene prednosti uporabe.

Prednosti samo-gostovanih partnerskih sistemov:

- Običajno ne plačujemo mesečne najemnine, pač pa plačamo licenco in po možnosti posodobitve.
- Zahtevajo tehnično znanje za namestitev na svoje strežnike, kar pogosto predstavlja dodatno breme za upravljalca oz administratorja.
- Primerni so v primerih, ko ima uporabnik zelo specifične zahteve, ki jih je težko realizirati na generalnem nivoju, sistem pa zahteva veliko prilagoditev.

Prednosti upravljano-gostovanih partnerskih sistemov:

- Uporabniku ni potrebno skrbeti za namestitev in delovanje sistema.
- Ponavadi se za njihovo uporabo plačuje mesečno najemnino, pogosto pa tudi provizijo od prodaje.
- Navadno so enostavnejši za uporabo in hitrejši za implementacijo.
- Primerni so za enostavnejšo uporabo.

1.2 Pregled obstoječih partnerskih sistemov

1.2.1 Samo-gostovani partnerski sistemi

iDevAffiliate

iDevAffiliate [6] je star partnerski sistem, ki ga je leta 1999 zasnovalo podjetje iDevDirect. Ima širok razpon funkcionalnosti: od prodajnih orodij, poročil in naprednih nastavitev, ki omogočajo visoko stopnjo prilagodljivosti partnerskega sistema.

Prednosti:

- Fleksibilnost nastavitev.
- Plačamo enkrat.
- Relativno-enostavna konfiguracija.

Slabosti:

- Posodobitve so brezplačne samo do enega leta.
- Za dodatne funkcionalnosti moramo kupiti dodatke.

Post Affiliate Pro

Post Affiliate Pro [7] je glavna konkurenca iDevAffiliate sistemu. Je zelo dodelana celovita rešitev, ki omogoča ogromno prilagoditev, vendar zahteva več tehničnega znanja kot pa druge rešitve. S tem omogoča tudi reševanje problemov in prilagoditve, ki so pri večini ostalih sistemov nemogoče.

Prednosti:

- Izjemno visoka stopnja prilagoditev.
- Možnost implementacije na več različnih spletnih strani (znotraj ene namestitve).
- Dobra podpora za večnivojske partnerske programe.
- Fleksibilna možnost plačila (omogočajo licenco z enkratnim in mesečnim plačilom).
- Podprta z mobilno aplikacijo.

Slabosti:

- Zahteva visoko stopnjo tehničnega znanja.
- Relativno dolga pot učenja v primerjavi z drugimi sistemi.

1.2.2 Upravljano-gostovani partnerski sistemi

V diplomski nalogi se bomo osredotočili na primerjavo upravljano-gostovanih sistemov. Ti sistemi so tudi zaradi njihovih prednosti osrednji del diplomske naloge. Upravljano-gostovani sistemi so se začeli na spletu pojavljati v zadnjih letih vzporedno z razvojem SaaS (Software as a Service) poslovnega modela in pojavom zavedanja prednosti rešitev v računalniškem oblaku.

Ambassador

Ambassador [8] je verjetno najbolj razširjena in priljubljena rešitev partnerskega sistema v oblaku, ki omogoča vse osnovne funkcionalnosti partnerskega sistema in je relativno enostavna za implementacijo. Ambassador je nastal leta 2010, vendar se je dobro razširil šele v zadnjih dveh letih.

Prednosti:

- Enostavna integracija.
- Prijazen uporabniški vmesnik.
- Dobra podpora osnovnim partnerskim funkcionalnostim.

Slabosti:

- Visoka cena – mesečni paketi se začnejo pri 200 USD na mesec.
- Ozek nabor možnih dodatnih prilagoditev (še posebej glede na ceno).

1.3 Motivacija in pregled naloge

V uvodu smo spoznali prednosti in slabosti obeh vrst partnerskih sistemov. Glavni del naloge pa je posvečen razvoju partnerskega sistema, s katerim želimo odpraviti nekatere omejitve obstoječih rešitev (prekompleksno funkcionalnost za večino primerov, nefleksibilnost sistemov in visoke cene uporabe) in narediti korak naprej pri enostavnejši integraciji z obstoječimi sistemi.

Drugo poglavje obravnava načrt razvoja sistema. V poglavju bomo predstavili agilen razvoj [9] in metodologije, ki omogočajo učinkovit razvoj podobnih aplikacij ali programskih rešitev. Predstavili bomo tudi programski jezik Ruby, podporno okolje Ruby on Rails [10] in vse ostale tehnologije in orodja, ki bodo uporabljene pri realizaciji naše programske rešitve.

V tretjem poglavju bomo opisali konkretno izvedbo načrta. Izvedba bo predstavljena po delih, obenem pa bodo prikazane tudi različne iteracije in razvoj projekta po posameznih delih funkcionalnosti.

V četrtem poglavju bomo prikazali uporabo končnega izdelka in integracijo s ostalimi sistemi.

V zadnjem, petem poglavju zaključujemo s pridobljenimi izkušnjami, ugotovljenimi napakami in načrtovanimi popravki. Za konec bomo pregledali še različne možnosti za nadaljni razvoj, izboljšave na partnerskem sistemu, kot tudi tržni pristop, promocijo tovrstnega orodja na spletu ter učne lekcije.

Poglavje 2

Načrt

2.1 Metodologije razvoja

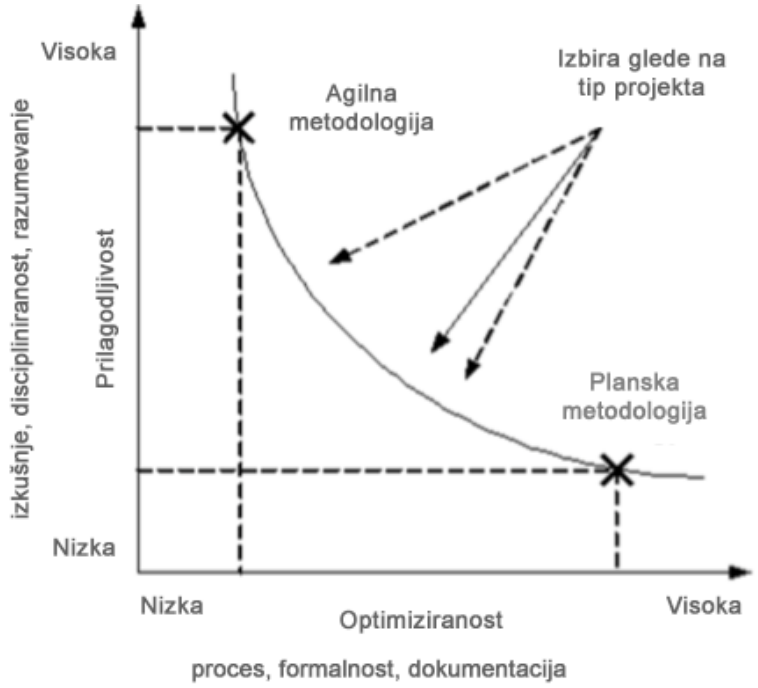
V tem delu bomo predstavili različne metodologije razvoja, po katerih smo se ravnali ob razvoju programske rešitve.

Metodologija [11] je skupek metod, principov in pravil, ki jih uporabimo na nekem področju z namenom regulacije postopkov. Določa jo analiza glavnega dela metod in principov povezanih z znanjem, ki je bilo pridobljeno na tem področju dela. Pri razvoju programske opreme metodologijo razvoja lahko definiramo kot razdelitev procesa razvoja v različne faze (korake), ki vsebujejo aktivnosti z nameom oziroma ciljem boljšega načrtovanja in upravljanja razvoja.

V glavni razdelitvi metodlogije glede na težavnost (slika 2.1) ločimo na agilne in planske metodologije [12].

Agilne metodologije uporabimo, kadar imamo odgovorne, disciplinirane in izkušene razvijalce, nepredvidljive in spreminjajoče zahteve ter stranko, ki je odprta za načine dela, ki zahtevajo aktivno sodelovanje in je pripravljena sodelovati pri razvoju programske rešitve.

Planske metodologije uporabimo, ko imamo opravka z relativno kompleksnimi sistemi in rešitvami, imamo manj izkušene razvijalce in ko imamo opravka z naročnikom, ki zahteva visoko stopnjo formalnosti (dokumentira-



Slika 2.1: Prikaz metodologij

nosti projekta).

2.1.1 Agilen pristop

Osnovni principi agilnih metod [9] so: prilagajanje in kontroliranje na konstantni bazi in filozofija, ki stremi k timskem delu in spodbujanju ter samoorganizaciji skupine oziroma posameznikov, ki delajo na posameznem projektu. Agilne metode so sestavljene iz najboljših praks, ki omogočajo hiter in učinkovit razvoj ter poslovni pristop, ki omogoča hitre in učinkovite prilagoditve rešitev končnemu uporabniku.

Agilne metode se značajsko precej razlikujejo od tradicionalnih načrtnousmerjenih, planskih metod razvoja [12]. Najizrazitejša razlika je, da se tovrstne metode fokusirajo k manj popolni dokumentaciji za dani problem in so osredotočene bolj na sam razvoj (pisanju kode ali ostalim aktivnostim, katerih rezultat je programska oprema).

Posledica takšne filozofije je, da pri razvoju projekta hitreje pridemo do uporabnih rezultatov in rešitev, se hitreje prilagodimo na dejanske probleme uporabnika (in trga), saj pri tovrstnem razvoju sodelujemo neposredno s končnim uporabnikom. Zahteve oziroma potrebe hitreje prilagodimo na bolj optimalen način kot pri tradicionalnih metodah razvoja. Zaradi narave in načel agilnih metodlogij so takšne metode primerne predvsem za manjše razvojne skupine, kjer lahko člani med seboj učinkovito komunicirajo in imajo več izkušenj.

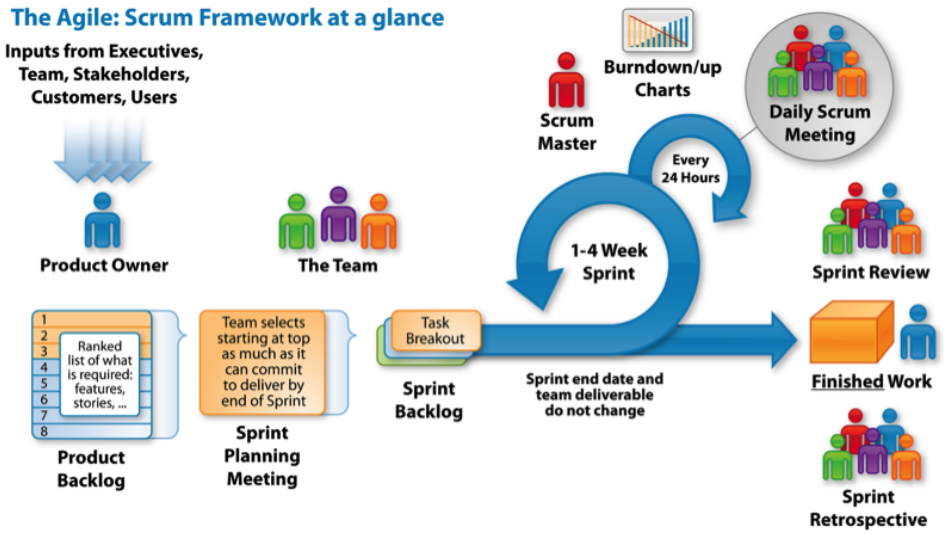
Osnovne vrednote agilnih metod, ki so jih avtorji definirali (manifest agilnih metodologij [14]) so:

- Posamezniki in interakcije so pomembnejše od procesov in orodij.
- Delujoča programska oprema je pomembnejša od popolne dokumentacije.
- Sodelovanje s stranko je pomembnejše od pogodbenih pogajanj.

- Odzivnost na spremembe je pomembnejša od togega sledenja načrtom.

Ob tem zapišimo še 12 principov agilnega razvoja [14]:

- Naša najvišja prioriteta je zadovoljiti stranko s hitrim in nepretrganim izdajanjem funkcionalnosti programske opreme.
- Sprejemamo spremembe zahtev, celo v poznih fazah razvoja. Agilni procesi vprežejo tovrstne spremembe v prid konkurenčnosti naše stranke.
- Delujočo programsko opremo izdajamo pogosto, znotraj obdobja nekaj tednov, do nekaj mesecev, s preferenco po krajšem časovnem okvirju.
- Poslovneži, naročniki in razvijalci morajo skozi celoten projekt dnevno sodelovati.
- Projekte gradimo okrog motiviranih posameznikov. Omogočimo jim delovno okolje, nudimo podporo in jim zaupamo, da bodo svoje delo opravili.
- Najboljša in najučinkovitejša metoda posredovanja informacij razvojni ekipi in znotraj ekipe same je pogovor iz oči v oči.
- Delujoča programska oprema je primarno merilo napredka.
- Agilni procesi promovirajo trajnostni razvoj. Naročniki, razvijalci in uporabniki morajo biti zmožni konstantnega tempa za nedoločen čas.
- Nenehna težnja k tehnični odličnosti in k dobremu načrtovanju izboljša agilnost.
- Preprostost – umetnost zmanjševanja količine nepotrebnega dela – je bistvena.
- Najboljše arhitekture, zahteve in načrti izhajajo iz tistih ekip, ki so samoorganizirane.



Slika 2.2: Prikaz metodologije Scrum

- V rednih časovnih razdobjih ekipa išče načine, kako postati učinkovitejša ob rednem prilagajanju svojega delovanja.

Obstaja veliko agilnih metodologij, vendar se bomo v diplomski nalogi osredotočili na Scrum metodologijo, saj je ta najbolj primerna in relevantna za tovrstne projekte in način razvoja.

2.1.2 Scrum

Scrum [15] je bolj znana agilna metodologija, ki se navadno uporablja pri upravljanju projekta oziroma programskega razvoja. Pogosto se Scrum dojema kot metodologijo, a v resnici je bolj podporno ogrodje pri upravljanju procesa razvoja programske rešitve [9].

Scrum definira fleksibilni in holistični pristop pri razvoju programske opreme, kjer razvojna ekipa deluje kot enota, da doseže določen zastavljeni cilj (slika 2.2). Metodologijo lahko razumemo kot odgovor na tradicionalne, sekvenčne metodologije in njihove predpostavke, ki so v določenih pogojih zelo okorne in neprimerne za doseganje učinkovitega razvoja.

Scrum spodbuja tesno in aktivno sodelovanje med vsemi člani ekipe [18], zato je metodologija primerna za manjše in izkušene ekipe. Ključni princip metodologije je zavedanje, da se med razvojnim procesom zahteve razvoja (oziroma zahteve uporabnikov) lahko spremenijo in da nepredvidljive situacije ne morejo vedno biti predvidene in pričakovane, kot predvidevajo tradicionalne planske razvojne metode, ki se zaradi tega bolj posvečajo pripravi dokumentacije.

Scrum zagovarja empirični pristop, ki sprejema predpostavko, da problem ne more biti v celoti razumljen. Zato je bolj pomembno osredotočanje na maksimiziranje zmožnosti ekipe, da čimprej priskrbi osnovno funkcionalnost, na podlagi katere reagira na dodatne potrebne spremembe in razširitve, ki se skozi uporabo osnovne verzije ob aktivnem sodelovanju končnega uporabnika pokažejo.

Zgradba

Projekti, ki jih vodimo po Scrum metodologiji napredujejo skozi zaporedje delov, ki jih imenujemo sprinti ali interacije [18]. Običajno je priporočena dolžina trajanja sprinta konstantne dolžine in navadno traja od 1 do 4 tedne ali največ 1 mesec. Cilj vsakega sprinta je načrtovan, izdelan in prezkušen izdelek oziroma njegov del.

Navadno med sprintom ne sprejemamo sprememb. Če želimo spremembe na projekt pogosteje uvajati, uvedemo za delo na projektu krajše sprinte, ki nam omogočajo krajše povratne zanke dela.

Vloge

V Scrum metodologiji dodelimo vloge osebam, ki sodelujejo pri razvoju. Scrum ponavadi vsebuje 3 glavne vloge. Te vloge so:

Lastnik produkta predstavlja investitorje in ostale lastnike, naročnike produkta. Je glas uporabnikov ter odgovorna oseba za opredelitev lastnosti izdelka. Skrbi, da ekipa z razvojem rešitve ustvarja ROI (Return On

Investment). Običajno lastnik produkta napiše uporabniške zgodbe (zahteve). Lastnik produkta je eden in se načeloma naj ne bi vpletal v delo ekipe oziroma tehnični razvoj, saj je to vloga Scrum vodje.

Razvojna ekipa je odgovorna za dostavo delov produkta (angl. P.S.I.) na koncu vsakega sprinta. Ekipo navadno sestavlja od 3 do 9 posameznikov, ki opravljajo konkretno delo (dizajn, programiranje, načrtovanje, testiranje, komunikacija ipd.). Ekipe so samo-organizirane, z vsemi znanji, ki jih delo na projektu zahteva.

Scrum vodja (angl. Scrum master) skrbi za izvedbo Scrum metodologije znotraj ekipe. Je oseba, ki je odgovorna za odpravljanje ovir za tekoče in nemoteno dostavljanje delov produkta in splošno delo na produktu. Vloga Scrum vodje ni samo tradicionalna vloga vodje projekta, pač pa ima bolj obširno vlogo, ki se razprostira od ustreznega izvajanja metodologije, do povezovanja članov ekipe, spodbujanja k izboljšanju in skrbi za zmanjšanje motečih dejavnikov na ekipo.

Sestanki

Ključni del metodologije so sestanki, ki imajo različne namene:

Planiranje sprinta obsega določanje prioritet z vidika analize in ocene dnevnika izdelka. V sklopu planiranja sprinta se odloči, kako želimo doseči cilje sprinta (planiranje) skozi dnevnik sprinta, ki obsega opravila, pri čemer izhajamo iz uporabniških zgodb (zahtev). V tem delu predvidimo trajanje dnevnika sprinta (navadno v urah). Planiranje sprinta poteka tako, da razvojna ekipa izbere posamezne postavke dnevnika izdelka, ki jih bodo izdelali v izbranem sprintu.

Na podlagi teh postavk se izdela dnevnik sprinta, v katerem so identificirana posamezna opravila v časovnem obsegu od 1 do 16 ur. Dnevnik sprinta izdela razvojna ekipa s pomočjo Scrum vodje.

Dnevni sestanek je namenjen hitremu obveščanju o poteku dela med člani razvojne ekipe in pomaga, da člani ekipe dobijo občutek, kaj je od

prejšnjega sestanka bilo narejeno. Glavni predmet tega sestanka je pogovor o tem, kaj je vsak član naredil včeraj, kaj planira za danes in hkrati informiranje o doseganju ciljev ter morebitnih težavah, s katerimi se posamezni član sreča. Na dnevnem sestanku se lahko dopolni dnevnik sprinta in navadno traja do 15 minut.

Retrospektiva sprinta je kritično samo-ocenjevanje skupine in preverjanje tega kaj deluje in kaj ne. Na retrospektivi sodeluje celotna razvojna ekipa in se opravi na koncu vsakega sprinta. Ponavadi traja od 15 do 30 minut, ekipa pa se na njem pogovori tudi o tem, katero delo bi radi pričeli, prenehali ali nadaljevali naprej.

Pregled sprinta je namenjen temu, da razvojna ekipa predstavi, kaj je bilo doseženo med sprintom. Gre za neformalno predstavitev, katere priprava lahko traja navadno do 2 uri in poteka brez prosojnic. Sestanek je namenjen vsem udeležencem, katerim se navadno v obliki demonstracije prikaže nove lastnosti, arhitekturo oziroma posodobitve, ki so bile opravljene na nekem izdelku.

Prilagojen pristop

V diplomski nalogi smo uporabili prilagojeno verzijo Scrum metodologije, saj smo programsko rešitev razvijali po lastnih zahtevah, zato smo imel hkratno vlogo lastnika produkta, razvojne ekipe in Scrum vodje. Projekt smo vodili sami, sledili pa smo ostalim praksam in priporočilom vodenja projekta po Scrum metodologiji [13].

Preden smo projekt začeli razvijati, smo napisali vse uporabniške zgodbe. Te zajemajo opis funkcionalnosti programske opreme iz vidikov vseh uporabniških vlog, ki bodo sistem uporabljale. Te so: promotor, lastnik izdelka, administrator.

Posamezne uporabniške zgodbe smo razdelili v sprinte in jih ustrezno ovrednotili glede na število ur pričakovanega dela na posamezni uporabniški zgodbi.

Pri delu in vodenju projekta smo si pomagali s orodjem Pivotal Tracker [16], ki omogoča vodenje in delo na projektu po metodologiji Scrum. Orodje omogoča zapis uporabniških zgodb in razporeditev v 3 glavne segmente:

- *Current*,
- *Backlog*,
- *Icebox*.

Current obsega trenutne zgodbe, ki so bile narejene in čakajo potrditev iz strani lastnika produkta (čigar vlogo sem imel jaz), *Backlog* obsega zgodbe, na katerih bomo začeli delati v nadaljevanju dela na projektu, *Icebox* pa obsega zgodbe, katere so pomembne za nadaljni razvoj in niso ključnega pomena v danem trenutku.

2.1.3 Testno usmerjen razvoj

Testno usmerjen razvoj (angl. test driven development) [17] je način razvoja programske opreme, ki temelji na programiranju testa (testov), ki opisuje novo funkcionalnost (zahtevo) ali določeno izboljšavo, nato pa se šele napiše minimalno količino kode rešitve, ki je potrebna, da je test uspešen. Na koncu ta del kode programer izboljša tako, da ustreza standardom kodiranja. Ta način razvoja naj bi spodbujal enostavne vzorce pisanja kode in samozavest razvijalca.

2.1.4 Uporabniške zgodbe

V začetni fazi projekta smo opisali uporabniške zgodbe, kot jih priporoča Scrum metodologija [19]. Uporabniške zgodbe za našo rešitev zajemajo 3 različne vloge uporabnikov:

Lastnik izdelka je uporabnik, ki storitev oziroma partnerski sistem uporablja za upravljanje in sledenje svojih promotorjev ter celotne statistike promocij svojih partnerjev.

Promotor je uporabnik, ki sistem uporablja zato, da lahko spremlja svoje prodaje, izplačila in druge dejavnosti, kot so dostop do informacij.

Administrator je uporabnik, ki skrbi in upravlja sistem ter ima pregled nad vsemi uporabniki (lastniki izdelkov in promotorji).

Promotor (angl. affiliate)

Uporabniške zahteve:

- Kot promotor se lahko registriram na spletni strani.
- Kot promotor lahko ustvarim novo promocijo za določen izdelek.
- Kot promotor lahko dobim promocijsko povezavo za določen izdelek.
- Kot promotor lahko promocijsko povezavo pošljem.

Lastnik izdelka

Uporabniške zahteve:

- Kot lastnik izdelka se lahko registriram na spletni strani.
- Kot lastnik izdelka lahko v sistem dodam nov izdelek.
- Kot lastnik izdelka lahko za dodan nov izdelek dobim kodo za enostavno spremljanje statistike na spletni strani, ki jo dodam v nogo HTML strukture.
- Kot lastnik izdelka lahko za določen izdelek dobim API žeton in skrivno kodo, katero uporabim na svoji platformi za klicanje API funkcij poročanja prodaj.
- Kot lastnik izdelka lahko izdelku določim privzeto provizijo za vsako prodajo izdelka.

- Kot lastnik izdelka lahko posameznemu uporabniku prilagodim provizijo na prodan izdelek.
- Kot lastnik izdelka lahko spremljam količino prodanih izdelkov v časovnem obdobju.
- Kot lastnik izdelka lahko spremljam količino klikov poslanih na prodajno stran izdelka.
- Kot lastnik izdelka lahko spremljam količino registracij za posamezni izdelek (nekončanih nakupov).
- Kot lastnik izdelka lahko spremljam profit vseh promotorjev na posamezen izdelek.
- Kot lastnik izdelka, lahko označim, katere prodaje so bile izplačane in kakšen je bil znesek izplačil promotorjem.

Administrator

Uporabniške zahteve:

- Kot administrator lahko dostopam na administracijsko kontrolno ploščo.
- Kot administrator lahko vidim statistiko registriranih lastnikov izdelka in njihovih promotorjev.
- Kot administrator lahko urejam račune vseh ostalih registriranih uporabnikov (lastnikov izdelka in promotorjev).
- Kot administrator lahko urejam uporabnike in njihove uporabniške podatke.

2.2 Uporabljene tehnologije

V tem poglavju bomo opredelili tehnologije (orodja in programske jezike), katere smo se odločili uporabiti za razvoj sistema.

2.2.1 Ruby on Rails

Ruby on Rails [10] je glavni programski jezik, ki smo ga uporabili na projektu. Ruby on Rails je ogrodje (angl. framework), ki temelji na programskem jeziku Ruby. Jezik je odličen iz vrste razlogov:

- Ima zelo lepo in ekspresivno sintakso.
- Predpisuje uporabo najboljših praks.
- Temelji na sistemu komponent (angl. gems).
- Omogoča hiter razvoj.

2.2.2 nGinx, Postgresql, Puma

Ostale tehnologije oziroma programska oprema, ki smo jo uporabili na projektu je sledeča:

nGinx

Za servisiranje spletnih vsebin oziroma zahtevkov potrebujemo programsko opremo spletnega strežnika. Najbolj znana je programska oprema Apache, vendar smo se za ta projekt odločil uporabiti nGinx [20], ki je zaradi hitrosti in kompatibilnosti z Ruby on Rails boljši.

Postgresql

Za bazo bomo uporabili objektno-relacijsko bazo Postgresql [27]. Za to bazo smo se odločili, ker je za tovrstni projekt najbolj primerna, saj deluje hitro v primerjavi z drugimi konkurenčnimi rešitvami in hkrati omogoča podporo različnim podatkovnim tipom (recimo Hstore, Json, Array) [28], ki nam delo olajšajo.

Ime entitete	Ime objekta	Opis
Uporabnik	User	Opisuje splošnega uporabnika
Izdelek	Product	Opisuje promocijski izdelek
Promocija	Promotion	Opisuje posamezno promocijo
Provizija	Commission	Opisuje provizijo pri prodaji
Analitika	Analytic	Opisuje statistiko določenega dneva
Izplačilo	Payout	Opisuje izplačilo provizije
Administrator	AdminUser	Opisuje administratorja
Obvestilo	Ipn	Opisuje dnevnik API zahtev

Tabela 2.1: Entitete uporabljene na projektu

Puma

Za serviranje Ruby aplikacije programski opremi nGinx potrebujemo tudi Ruby strežnik. Za rešitev smo uporabili strežnik Puma [21], ki pride v paketu komponent Ruby-a. Za Pumo smo se odločili, ker je najhitrejši Ruby strežnik, podpira pa tudi delo z nitmi (angl. threads).

2.3 Struktura baze

Pri strukturi baze smo opredelili, katere entitete, attribute in podatkovne tipe ter relacije potrebujemo za realizacijo opisane funkcionalnosti.

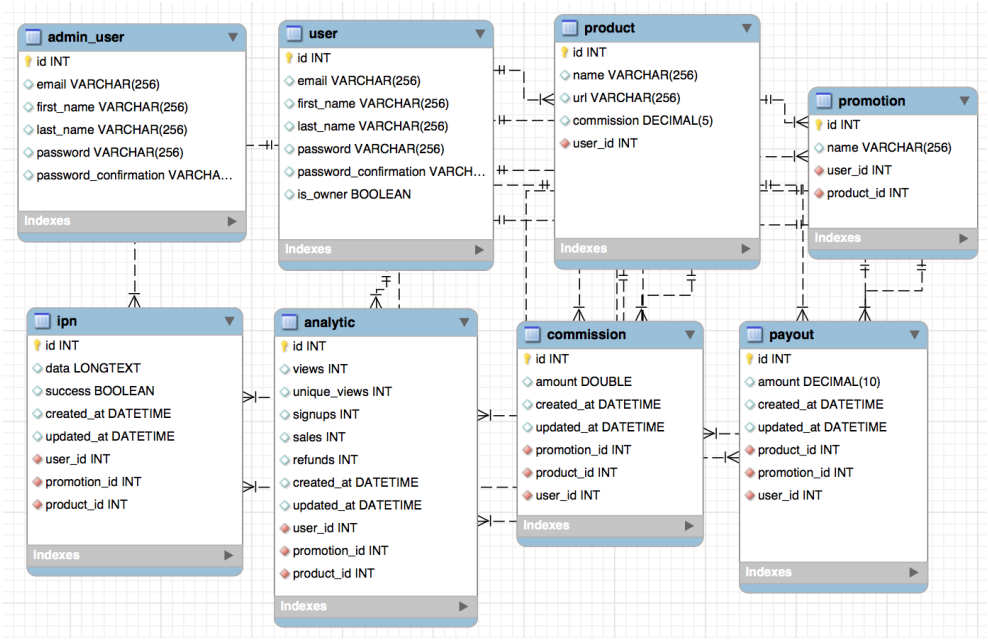
Entitete, ki smo jih uporabili, so prikazane v tabeli 2.1.

Prikaz vseh relacij, atributov in podatkovnih tipov je podan na sliki 2.3.

2.4 Integracije z zunanjimi orodji

V tem delu bomo opredelili delovanje sistema v okviru povezovanja z zunanjimi orodji in sistemi. Za boljšo preglednost je to poglavje razdeljeno na dva dela:

- Integracija s sistemi za vzpostavitev partnerskega sistema, ki zajema



Slika 2.3: Končni ER diagram

razumevanje, kako naš partnerski sistem fukcionira v sklopu povezovanja z ostalimi orodji, znotraj katerih ga lahko uporabimo.

- Integraciji z zunanjima orodjema za lažji vpogled v delovanje sistema, ki nam pomagata pri učinkovitem spremljanju aplikacije za spremljanje učinkovitosti delovanja in spremljanja uporabnikov (analitike).

2.4.1 Integracija s sistemi za vzpostavitev partnerskega sistema

Ideja našega partnerskega sistema je, da ga lahko enostavno integriramo v obstoječe rešitve, predvsem SaaS aplikacije in spletne trgovine (angl. e-commerce).

Integracija je sestavljena iz dveh delov [22]:

- Integracija na aplikacijski strani (angl. frontend)

- Integracija na strežniški strani (angl. backend)

Integracija na aplikacijski strani poteka tako, da na njo namestimo Javascript kodo, ki izvrši posodabljanje statistike o obiskih, napotitvah in namestitvi piškotka na stran uporabnika, da ga lahko ustrezno zaznamo in njegovo registracijo oziroma nakup pripišemo ustreznemu promotorju ob končni akciji.

Integracija na strani strežnika je integracija, ki jo navadno napravimo znotraj kode, ki se izvaja na strani strežnika oziroma bolj poredko na strani odjemalca (brskalnika). Ta del definira API klice ob dveh ključnih akcijah: ko uporabnik opravi registracijo in ko uporabnik kupi poljuben izdelek.

Ob koncu projekta bo za lažjo integracijo na strani strežnika napisana tudi Ruby knjižica, ki se za še hitrejšo integracijo lahko enostavno vključi v projekt in uporabi njene že napisane klice.

2.4.2 Integraciji z zunanjima orodjema za lažji vpogled v delovanje sistema

Za lažje spremljanje dogajanja na aplikaciji in spremljanja uporabnikov želimo uporabiti dve ključni orodji, ki nam bosta olajšali delo in vpogled v delovanje.

AppSignals

AppSignals [23] je storitev v oblaku, ki nam omogoča enostavno spremljanje napak v aplikaciji. V primeru, da se na aplikaciji zgodi neka napaka, se ta pošlje v naš AppSignal račun in zabeleži s čimveč podatki in kontekstu te napake. Tako je spremljanje napak in njihovo odpravljanje dosti lažje. Poleg spremljanja napak pa nam to orodje omogoča tudi enostavni pregled nad hitrostjo in učinkovitostjo dela aplikacije (hitrost delovanja strežnika in čas posameznega zahtevka), kar nam omogoča, da lahko kakšne nepravilnosti pravočasno zaznamo in odpravimo.

Mixpanel

Mixpanel [5] je napredna analitična rešitev, ki nam omogoča spremljanje oziroma sledenje uporabnikom in njihovemu obnašanju. Z njim lahko enostavno vidimo, kako se posamezen uporabnik vede znotraj naše aplikacije in kaj so njegove značilnosti. Sami lahko definiramo različne parametre in vrednosti, ki so za nas pomembne pri uporabi določene aplikacije; v našem primeru je to: tip uporabnik, število promocij, število prodaj, vsota zaslužkov ipd.

Mixpanel nam poleg tega omogoča tudi avtomatizacijo promocijskih aktivnosti. Tako lahko recimo uporabniku, ki se že več kot 10 dni ni vpisal v aplikacijo in še ni ustvaril promocije, enostavno pošljemo elektronsko sporočilo in ga nagovorimo naj to opravi. S tem povečamo možnost učinkovite vpeljave uporabnika v našo aplikacijo.

Poglavje 3

Razvoj

V tem poglavju smo prikazali, kako je potekal razvoj partnerskega sistema. V prvem delu bomo predstavili ključne funkcionalnosti aplikacije, nato bomo prikazali, kako se je ta funkcionalnost razvijala skozi sklope razvoja, na način, ki smo ga predstavil znotraj Scrum metodologije.

Na koncu bomo pogledali, kako je potekala postavitev strežnika in namestitev na produkcijsko okolje.

3.1 Razvoj funkcionalnosti sistema

Začetek razvoja je bil zasnovan na posameznih glavnih sklopih funkcionalnosti, katere lahko razdelimo na 3 ključne dele:

3.1.1 Upravljanje uporabnikov

Prva ključna funkcionalnost je upravljanje uporabnikov, ki zajema registracijo uporabnika (lastnika izdelka oziroma promotorja) ter v ozadju administracijo vseh teh uporabnikov.

Po tem, ko je bila znana struktura in zasnova podatkovne baze (relacije in vsi potrebni podatki), smo pričeli z implementacijo logike za registracijo in prijavo uporabnikov. Za Ruby on Rails obstaja komponenta imenovana Devise [24], ki omogoča poenostavljeno implementacijo uporabniške logike za

registracijo in prijavo v aplikacijo. Devise podpira zelo širok nabor naprednih funkcionalnosti (kot so oAuth [26], integracije z zunanjimi avtentikacijskimi skrbniki), vendar smo v našem primeru potrebovali le osnovno funkcionalnost.

Zatem smo namestili še paket ActiveAdmin [25], ki omogoča hitro in učinkovito postavitev administracijskega vmesnika.

Sledila je osnovna konfiguracija in prilagoditev obeh dodatkov glede na našo željeno funkcionalnost.

Devise smo prilagodili tako, da sprejme vsa dodatna polja, ki jih pri našem sistemu potrebujemo (ime, priimek, PayPal naslov za plačila ipd.) ter ActiveAdmin prilagodil tako, da omogoča pregled nad vsemi uporabniki in njihovo urejanje.

3.1.2 Spremljanje statistik

Druga ključna funkcionalnost je bila logika okoli funkcionalnosti spremljanja statistik.

Odločili smo se za zasnovo, ki minimizira podvojene zapise v bazi in omogoča učinkovit pregled nad dnevnimi statistikami.

Vsaka promocija ima asociacijo na analitiko (entiteta Analytic), ki v posameznem zapisu definira število ogledov, unikatnih ogledov, vpisov in prodaj za posamezen dan. Tako imamo podatke urejene po dnevih, kar nam omogoča osnovni pregled za dogajanje nad statistikami.

Nad zapisi lahko enostavno izvajamo matematične operacije (seštevke, povprečja ipd.) v posameznih obdobjih.

3.1.3 API Integracije

V strežniškem delu aplikacije želimo imeti možnost API klicev, ki nam omogočajo, da posamezne klice zapisovanja statistik lahko kličemo poljubno glede na željen dogodek v sistemu, ki ga povezujemo z našim partnerskim sistemom.

Tako sta osnovni funkciji, ki jih želimo imeti:

- Track event – zabeleži posamezen dogodek, kot je recimo registracija (angl. signup) ali ogled (angl. view).

- Track charge – zabeleži nakup izdelka.

3.2 Pregled iteracij razvoja

V tem delu bom predstavil posamezne interacije (sprinte) za razvoj projekta in jih razgradil glede na potek in spremembe, ki so sledile, ko sem naletel na določene težave in probleme.

Za beta različico projekta sem planiral 6 iteracij, za katere sem zaradi večje agilnosti predvidel povprečno dolžino enega tedna [13]. Ker sem na projektu delal sam, so nekateri sprinti potekali malenkost drugače, kot pa bi v primeru, če bi jih izvajali znotraj ekipe v standardni obliki, saj sem sam izvajal tudi vlogo Scrum masterja in lastnika produkta.

Seznam iteracij (tabela 3.1) je predstavljen na način, da v levem stolpcu opredeli planirano delo za to obdobje, v nadaljevanju pa so opisane spremembe oziroma nepričakovane stvari, ki so vplivale na dejanski razvoj projekta.

V nadaljevanju podajam opis posameznih interacij.

3.2.1 Teden 1: Priprava na razvoj, implementacija uporabniške prijave

V prvem tednu sem se osredotočil na pripravo okolja in pripravo podatkovne baze ter modelov za delo na projektu. Priprava okolja je zajemala namestitev in kreiranje novega Rails projekta, kateremu sem namestil vse osnovne komponente in programsko opremo, ki je potrebna za razvoj. Poleg komponent (angl. gems) sem namestil tudi Postgresql bazo, za katero sem se odločil zaradi dobre podprtosti na vseh sistemih ter možnostjo zapisa podatkovnega tipa zgoščene tabele (hStore) [28].

Št.	**Plan iteracije**	**Spremembe / opombe**
1	• Vzpostavitev okolja. • Definiranje strukture baze in razrednih modelov. • Implementacija uporabniške prijave.	
2	• Pisanje osnovne logike promocij. • Implementacija adminstratorskega vmesnika.	
3	• Delo na osnovnem uporabniškem vmesniku. • Dodajanje vlog uporabnikov (razlike med vlogami).	• Implementacija uporabniškega dela je zahtevala nekaj sprememb na *device* komponenti.
4	• Implementacija API klicev. • Delo na Javascript kodi za namestitev na uporabniški del.	
5	• Integracija z Mixpanel in Appsignals. • Sprememba strežnika na Puma komponento (angl. gem). • Delo na službah za procesiranje v ozadju (komponenta *sidekiq*). • Delo na analitiki za administratorja.	• Aktivno delo na izboljšavah in testiranje.
6	• Priprava produkcijskega strežnika. • Priprava skript za namestitev aplikacije na strežnik (angl. deploy). • Možnost dodajanja izdelkov za testiranje na testnem okolju.	
7	• Integracija partnerskega sistema v aplikacijo RankTrackr. • Testiranje sistema na produkcijskem okolju.	

Tabela 3.1: Seznam iteracij

Sledilo je pisanje migracij, ki nam v Ruby on Rails služijo definiranju baze: podatkovnih tipov, relacij in atributov modelov. Posebnost migracij je, da nam omogočajo enostavno spreminjanje arhitekture baze glede na fazo projekta [29]. Tako lahko strukturo in razvoj baze enostavno spremljamo in premaknemo tudi nazaj v čas, kar je zelo koristno za agilni razvoj, saj omogoča veliko fleksibilnosti.

Postavili smo tudi osnovo za uporabniško prijavo. Uporabili smo zelo poznano komponento imenovano Devise, ki nam delo poenostavi in omogoča veliko dodatne funkcionalnosti. V našem primeru potrebujemo le enostavno registracijo in prijavo z identifikacijo uporabnika preko e-poštnega naslova. Poleg osnovnih podatkov o uporabniku smo morali naredili nekaj prilagoditev na komponenti, tako da sprejme vse podatke, ki jih o posameznem uporabniku potrebujemo.

3.2.2 Teden 2: Logika modelov in implementacija administracijskega vmesnika

Logika modelov

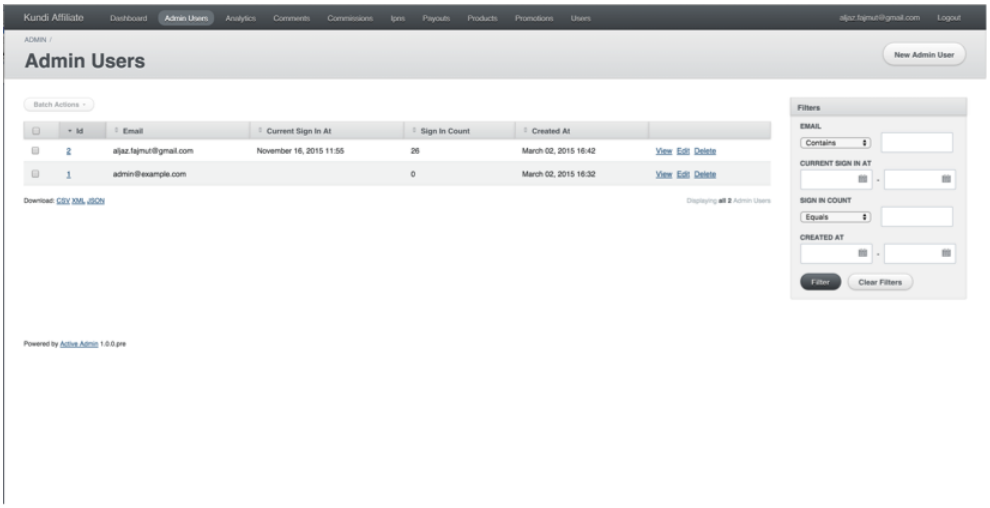
V drugem tednu smo se posvetili logiki modelov. To delo je zajemalo definiranje metod znotraj modelov, ki jih aplikacija uporablja (primer je na sliki 3.1). Poleg metod smo definirali tudi validacijske nastavitve za zapisovanje v bazo, relacije in obnašanje modelov v različnih kontekstih, ko se instanca na novo ustvari ali zapiše v bazo.

Administracijski vmesnik

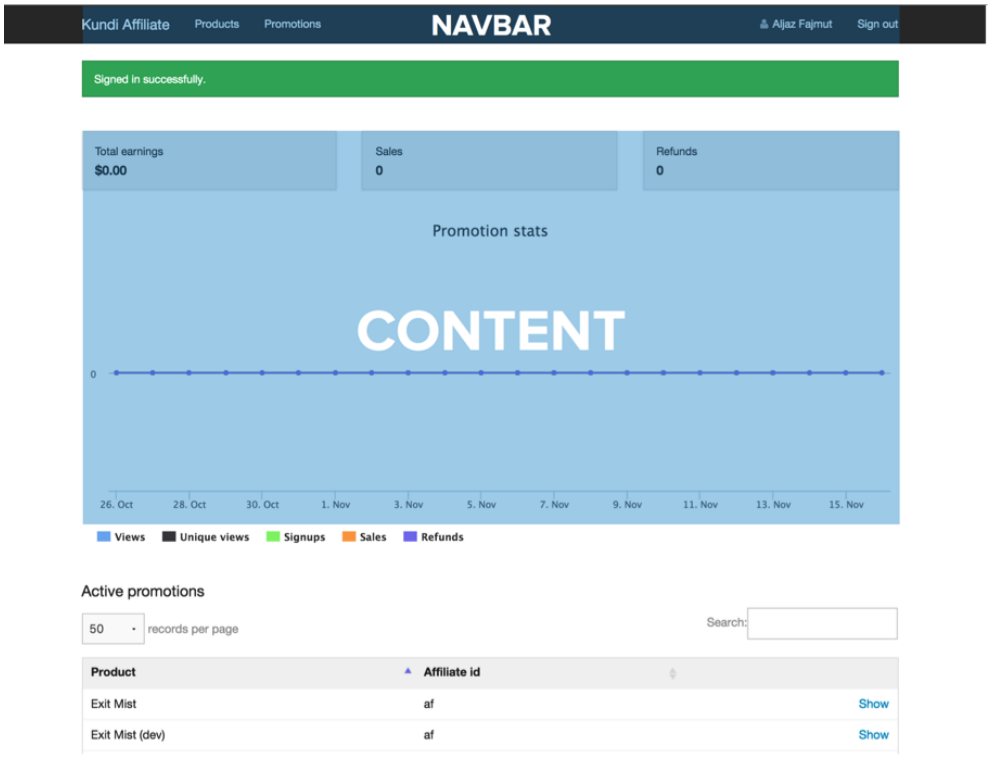
S pomočjo komponente Activeadmin smo pripravili osnovo za administracijski vmesnik (slika 3.2). Definirali smo vidnost modelov, ki jih uporabljamo – osnovne podatke, ki jih želimo vidne na nivoju zbirke in posameznega zapisa uporabnika (User), provizije (Commission), produkta (Product) in promocije (Promotion).

```ruby
class Commission < ActiveRecord::Base
  belongs_to :user
  belongs_to :product
  belongs_to :promotion
  belongs_to :ipn

  validates :charge_amount, presence: true
  validates :promotion, presence: true

  before_save :check_relations
  before_save :calculate_commission

  after_create :track_analytics

  CHARGE_TYPES = [ 'sale', 'refund' ]

  scope :sales, -> { where(charge_type: 'sale') }
  scope :refunds, -> { where(charge_type: 'refund') }

private
```

Slika 3.1: Logika razreda Provizija (angl. Commission)



Slika 3.2: Administracijski vmesnik



Slika 3.3: Uporabniški vmesnik

3.2.3 Teden 3: Uporabniški vmesnik, vloge uporabnikov

Uporabniški vmesnik

Pripravili smo tudi predlogo za uporabniški vmesnik (slika 6). Uporabili smo komponento Foundation [30], ki zajema skupek HTML in CSS elementov ter komponent za uporabniški vmesnik. Na osnovni knjižice smo definirali osnovno postavitev elementov in izgled aplikacije.

Razlikovanje uporabnikov

Ker želimo omogočati različnim uporabnikom različne funkcionalnosti (recimo lastnik produkta mora imeti možnost urejanja izdelkov in pregled nad promotorji) je bilo potrebno uporabniški prijavi prirediti nekaj logike. Odločili

smo se, da v tabelo uporabnikov dodamo nov atribut `product_owner`, ki loči med promotorji in lastniki izdelkov. Za administratorja smo se odločili za ločevanje na drugem nivoju in za njega obstaja poseben razred AdminUser, saj je tovrstno razlikovanje najbolj smiselno (administrator je nad ostalimi uporabniki in je redko v drugih vlogah).

3.2.4 Teden 4: API funkcionalnost, Javascript koda za uporabniško stran

API

V tem tednu smo pozornost posvetili implementaciji API funkcionalnosti. Pri tem smo se držali priporočil in dobrih REST praks [31], tako da smo te definirali znotraj naslova `/api/v1/<posamezen klic>`.

Klici omogočajo osnovne funkcionalnosti beleženja dogodkov: `track_event` za beleženje posameznih akcij (ogled, registracija) in `track_charge` za beleženje posameznih nakupov izdelkov.

Za avtentikacijo je potrebna uporaba skrivnega ključa, ki ga uporabnik dobi potem, ko doda nov izdelek v sistem. Za tovrsten način avtentikacije smo se odločili, ker je enostaven in učinkovit ter najbolj primeren glede na naše preproste API zahteve.

Javascript koda

Za doseg ciljev projekta (enostavno uporabo in implementacijo v obstoječe rešitve) potrebujemo tudi Javascript kodo, ki jo uporabnik lahko enostavno namesti na spletno stran.

Funkcija te skripte je, da ko uporabnik pride prek promocijske povezave na npr. `http://mojatrgovina.com/?aff=xy`, ta vstavi piškot na njegov računalnik, ki zabeleži skrajšan niz promotorja (v tem primeru xy), ki je tega uporabnika napotil na to stran, hkrati pa se zabeleži ogled te strani. V primeru, da uporabnik s piškotom, ki je nameščen v njegov brskalnik

opravi nakup, sistem ustrezno pripiše vrednost nakupa promotorju, ki je tega uporabnika napotil na spletno stran in ustrezno izračuna provizijo.

Pričakovali smo, da bomo to skripto končali v enem tednu, vendar pa se je čas pisanja nekoliko zavlekel še v naslednji teden, saj se je izkazalo, da je zadevo bolj kompleksna – koda se mora poganjati v izolaciji, saj nočemo, da v kolikor na strani pride do napake v drugi nameščeni skripti, da ta napaka vpliva na delovanje naše knjižice. Poleg tega za pomoč uporabljamo knjižico jQuery [32], ki določene klice poenostavi. Posebej pozorni moramo biti v primeru, če je jQuery že nameščen na spletno stran, kar je zelo pogosto. V tem primeru je potrebno ustrezno izolirati našo instanco od jQuery-a [33].

3.2.5 Teden 5: Integracije, Puma, procesiranje v ozadju, analitika za administratorja

Integracije

V tem tednu smo se posvetili integraciji zunanjih rešitev – Mixpanel in Appsignals. Za integracijo s Mixpanel smo za osnovo uporabili komponento Mixpanel, ki nam omogoča enostavno inicializacijo objekta preko katerega kličemo posamezne dogodke (angl. events) in nastavljano uporabniške lastnosti. Posebnost te komponente je, da uporablja Rails rack, kar nam omogoča, da pred izvršitvijo funkcije kontrolerja nastavimo, kateri dogodek želimo v Mixpanel poslati in ta poskrbi, da se ustrezna Javascript koda vstavi v stran, ki je prikazana uporabniku.

Vsaka akcija uporabnika znotraj aplikacije je prek kontrolerjev ustrezno posredovana na Mixpanel, kar nam omogoča enostavno sledenje njihovega obnašanja in načinov uporabe.

Na drugi strani nam integracija s storitvijo Appsignals omogoča, da se vse napake, ki se zgodijo na strežniku ustrezno zabeležijo z vsemi podatki na Appsignals strežniku, kar nam olajša spremljanje in sledenje težavam ter učinkovito odpravo napak.

Puma

Na začetku projekta smo za osnovni vmesnik s nGinx strežnikom uporabljali komponento Unicorn, vendar pa smo po prebiranju literature in lastnih ugotovitvah odkrili, da ni tako učinkovit v določenih scenarijih, saj ne podpira niti. Zato smo ta vmesnik zamenjali s komponento Pumo, ki je novejši produkt Rails skupnosti in pridobiva veliko pozornosti.

Procesiranje v ozadju

Upoštevajoč dejstva, da določena akcija, odvisna od drugih zunanjih strežnikov, lahko zaradi odvisnosti traja dlje kot smo pričakovali (ali pa se zaradi nedosegljivosti strežnika sploh ne izvrši uspešno), želimo vpliv teh akcij na uporabniško izkušnjo in stabilnost sistema zmanjšati, zato smo se za rešitev problema odločili uporabiti procesiranje v ozadju.

Primer, ko potrebujemo takšno obvladovanje situacije: ko na strežniški strani (v kontekstu API dostopa, kjer ne moremo izvrševati Javascript klicev na uporabniški strani) kličemo Mixpanel API iz kontrolerja oziroma razreda aplikacije, ker želimo uporabniku posodobiti določeno lastnost.

Uporabili smo odprtokodno rešitev Redis [34], ki je spominska podatkovna struktura, ki jo lahko uporabljamo za medprocesno komuniciranje ter komponento Sidekiq [35], ki nam omogoča lažjo implementacijo procesiranja opravil v ozadju.

Takšna rešitev nam omogoča, da lahko statične metode na razredih kličemo z metodo delay, ki klic metode preda v izvrševanje v ozadje. V primeru, da se klic ne zaključi uspešno (javi izjemo), se klic ponovi maksimalno 3×, vsakič v daljšem intervalu.

Del kode, kjer uporabljamo takšen pristop je v modelu uporabnika `User`, prikazan na sliki 3.4.

```ruby
def track_charge
  if params[:amount]
    @commission = @promotion.commissions.create!(
      # if commission amount
      amount: params[:commission_amount],

      # original amount + type
      charge_amount: params[:amount],
      charge_type: params[:charge_type],

      # save transaction id
      charge_id: params[:charge_id],

      # relations + data
      data: params[:data].to_h.inspect,
      ipn: @ipn
    )

    @ipn.update_attributes amount: params[:amount]

    User.delay.set_mixpanel_properties(@promotion.user.id)

    success
  else
    unprocessable_entity('no amount specified')
  end
end
```

Slika 3.4: Primer izvrševanja v ozadju

Analitika za administratorja

V administracijskem vmesniku želimo boljši pregled nad uporabo sistema, posameznih uporabnikov in promocijami. Prišli smo do elegantne rešitve, ki nam omogoča, da velik del kode abstraktno uporabimo na več delih, kjer potrebujemo podobne analitične povzetke.

Ruby on Rails omogoča poseben koncept imenovan Concerns (skrbništvo) [37], ki nam omogoča, da določeno funkcionalnost razreda na enostaven način razširimo s tem, da v mapo `/models/concerns` dodamo ta razred.

S tem namenom smo ustvarili skrbniški razred `hasAnalytics`, ki nam omogoča, da nad zapisi in tabelami lahko izvajamo enotne in konsistentne analitične akcije, kot so:

- Total between – seštevek rezultatov med dvema datumoma na posamezni celici.
- Period count – seštevek zapisov med dvema datumoma

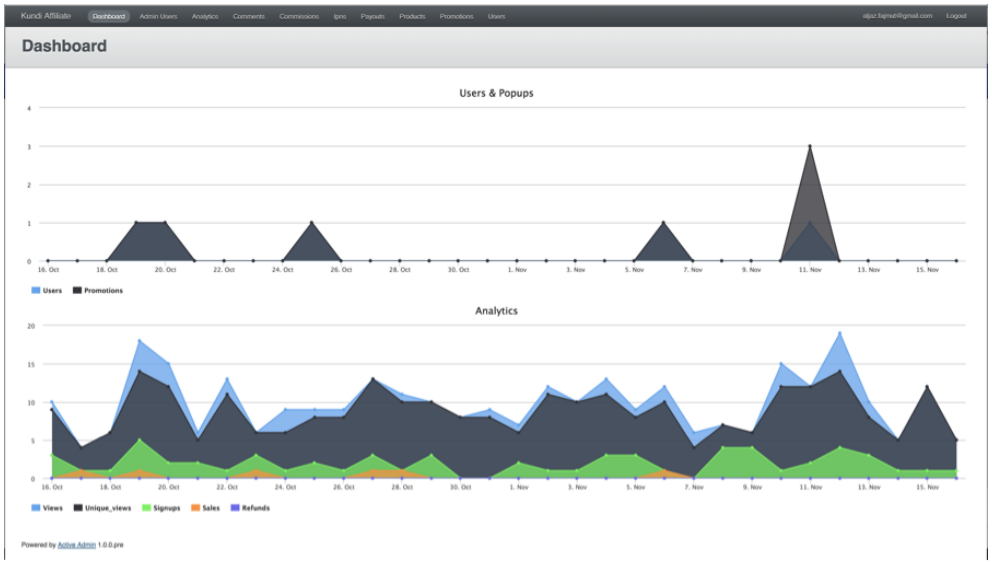
Ti dve funkciji uporabljata privzeta `created_at` in `updated_at` časovni polji (angl. timestamp), ki jih Ruby on Rails privzeto priredi vsakemu izmed razredov. S takšno abstrakcijo lahko enostavno razširimo uporabnost razreda. To dosežemo tako, da nad skupino zapisov posameznega razreda izvedemo matematične operacije, s čimer dobimo pregled ustvarjenih zapisov ali seštevkov določenih celic glede na določen datum.

Za prikaz grafov smo uporabili znano knjižico Highcharts [38], ki je zelo fleksibilna in omogoča vse potrebne funkcionalnosti za risanje grafov.

Tako lahko administrator enostavno vidi aktivnosti v sistemu v nadzorni plošči (slika 3.5).

3.2.6 Teden 6: Priprava produkcijskega okolja in skript za namestitev aplikacije

V tem tednu smo se osredotočili na pripravo strežnika za produkcijsko okolje ter skripte, ki nam bodo ob spremembah olajšale posodabljanje aplikacije in



Slika 3.5: Vizualizacija za administratorja

namestitev na strežniku.

Strežnik smo konfigurirali po priporočljivi Ruby on Rails VPS (Virtual Private Server) konfiguraciji in postavili na Linux Ubuntu operacijskem sistemu. Za namestitev oziroma uvajanje aplikacije si v Ruby on Rails pomagamo s skriptami za uvajanje.

Podrobnosti konfiguracije strežnika in skript za uvajanje smo predstavili v naslednjem podpoglavju Priprava na produkcijsko okolje.

3.2.7 Teden 7: Integracija sistema v aplikacijo RankTrackr, testiranje v produkcijskem okolju

V tem tednu smo se fokusirali na integracijo partnerskega sistema v obstoječo aplikacijo RankTrackr [39]. Ustvarili smo svoj uporabniški račun, dodali produkt RankTrackr in namestili Javascript kodo sistema na spletno stran RankTrackr-ja .

Potrebnih je bilo tudi nekaj sprememb v strežniškem delu aplikacije. Za lažjo integracijo smo napisali knjižico v Rubyu, ki omogoča enostavnejšo

integracijo v podobne aplikacije in nam ponuja vse potrebne metode, ki jih potrebujemo za integracijo.

Podrobnosti o testiranju na produkcijskem okolju so podane v poglavju 4.3 Test na produkcijskem okolju.

3.3 Priprava na produkcijsko okolje

3.3.1 Konfiguracija strežnika

Na strežniku teče Linux Ubuntu 14.04, saj je široko podprt operacijski sistem in ga podpira velika večina vseh VPS (Virtual Private Server) ponudnikov. Kot ponudnika gostovanja smo izbrali Digitalocean [41]. Zelo všeč nam je njihova platforma, ta omogoča enostavno prilagajanje lastnosti strežnika, ki ga potrebuješ (v primeru, da se potrebuje več RAM-a, ni potreben nakup novega strežnika in migracija vseh stvari, pač pa se strežnik samo ugasne in počaka slabo minuto, da se proces konča).

Na strežniku smo ustvarili novega uporabnika deploy, ki ga uporabljamo samo za dostop ob namestitvi in posodabljanju aplikacije. Na strežnik smo prav tako namestili rbenv [36] (okolje za upravljanje Ruby verzij na sistemu), nGinx strežnik [20], Postgresql bazo [27] in Redis [34].

3.3.2 Namestitev na strežnik

V Ruby on Rails za namestitev aplikacije na strežnik uporabljamo tako imenovane deploy skripte, ki nam poenostavijo delo. Za tovrstno funkcionalnost smo uporabili obstoječo komponento imenovano Capistrano [40]. Capistrano vsebuje nekatere že definirane skripte, kot so premik datotek (preverjanje zadnje različice na Git repozitoriju [42] in osvežitev zadnjih sprememb), namestitev kompontent (angl. gems), zagon / zaustavitev / ponovni zagon strežnika – Puma [21], nGinx [20].

Našo skripto sem priredil tako, da omogoča uvajanje aplikacije v 2 okolji: produkcijsko okolje in uprizoritveno okolje, katerega lahko uporabimo za to,

da določene funkcionalnosti testiramo na produkcijskem strežniku preden jih začnemo uporabljati v produkciji.

Naša osnovna skripta za nameščanje aplikacije (slika 3.6) je sestavljena iz naslednjih korakov:

- Preverjanje revizije iz Git-a [42] (preveri ali imamo zadnjo verzijo na Git strežniku).
- Zagon testov preveri, če se vsi testi pravilno izvedejo.
- Prenos datotek, namestitev kompontent, izvedba migracij (osnovni korak).
- Priprava datotek za nameščanje.
- Ponovni zagon aplikacije.
- Čiščenje začasnih datotek.

```
93  namespace :deploy do
94    # make sure we're deploying what we think we're deploying
95    before :deploy, "deploy:check_revision"
96    # only allow a deploy with passing tests to deployed
97    before :deploy, "deploy:run_tests"
98    # compile assets locally then rsync
99    # after 'deploy:symlink:shared', 'deploy:compile_assets_locally'
100   after :finishing, 'deploy:cleanup'
101
102   desc 'Restart application'
103   task :restart do
104     on roles(:app), in: :sequence, wait: 5 do
105       invoke 'puma:restart'
106     end
107   end
108
109   after :publishing, :restart
110
111   after :restart, :clear_cache do
112     on roles(:web), in: :groups, limit: 3, wait: 10 do
113       # Here we can do anything such as:
114       # within release_path do
115       #   execute :rake, 'cache:clear'
116       # end
117     end
118   end
```

Slika 3.6: Skripta za nameščanje aplikacije na strežnik

Poglavje 4

Uporaba

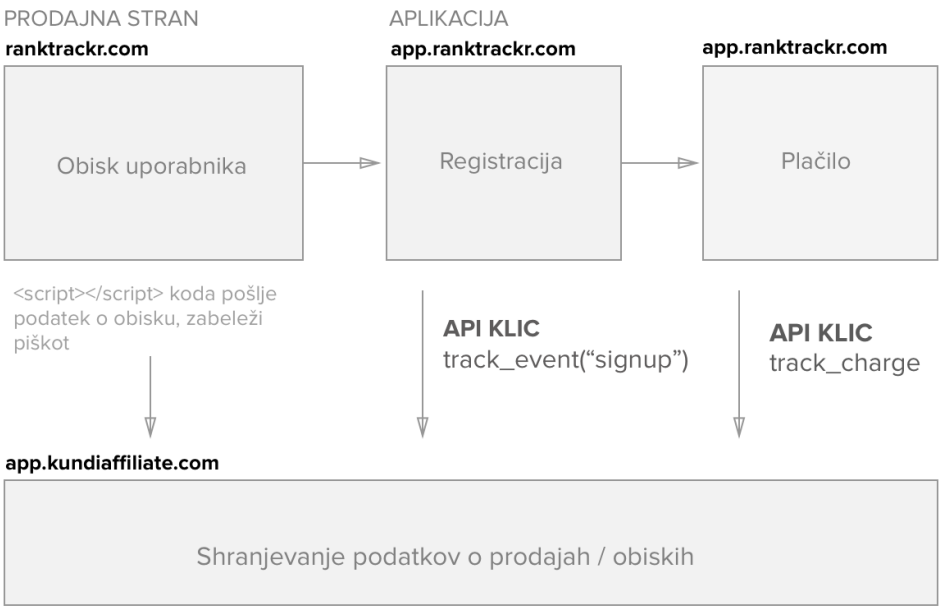
V tem poglavju bomo predstavili primer uporabe partnerskega sistema in njegovo integracijo v obstoječo rešitev.

4.1 Končni izdelek

Končni izdelek ponuja vso funkcionalnost, ki smo jo podali v zahtevah. Hkrati pa tudi enostavno integracijo v obstoječe platforme – tako v SaaS (Software as a Service) aplikacije, kot tudi e-trgovine in ostale rešitve.

4.2 Integracija v obstoječo rešitev

Cilj razvoja partnerskega sistema je bil uporaba v naših obstoječih aplikacijah, zato smo jo integrirali v aplikacijo RankTrackr [39], ki je namenjena spremljanju pozicije strani na spletnih iskalnikih (slika 4.1). Z opisano nadgradnjo smo dosegli, da našo obstoječo aplikacijo lahko promovirajo drugi uporabniki in za to dobijo ustrezno nagrado. Spodnja slika (slika 4.1) prikazuje delovanje aplikacije RankTrackr v povezavi z nadgradnjo z razvitim partnerskim sistemom. Na prodajni strani imamo nameščeno skripto, ki na uporabnikov računalnik namesti piškotek, na podlagi katerega lahko ugotovimo, kdo je uporabnika na našo spletno stran napotil. V kolikor se ta upo-



Slika 4.1: Shema delovanja sistema

rabnik registrira in kasneje plača storitev, se to prek API integracije sporoči v naš partnerski sistem in ustrezno določi provizija, ki se pripiše promoterju, ki je uporabnika napotil na našo storitev.

Poleg kreiranja novega projekta in dodajanja Javascript kode na uporabniški del spletne strani RankTrackr-ja, je bilo potrebnih tudi nekaj sprememb v strežniškem delu aplikacije. Za lažjo integracijo smo napisali knjižico v Rubyu, ki omogoča enostavnejšo integracijo v podobne aplikacije in nam ponuja vse potrebne metode za API klice, ki jih potrebujemo za integracijo.

V našem primeru smo knjižico uporabili na dveh koncih:

- Ko se uporabnik registrira na aplikaciji, kličemo metodo `track_event("signup")`, ki nam zabeleži registracijo uporabnika.

- Ko dobimo obvestilo od PayPal, da je uporabnik opravil nakup, kličemo metodo `track_charge`, kateri podamo znesek in identifikacijo promotorja, ki je uporabnika napotil na stran.

Prav tako smo morali prilagoditi del ob registraciji, da ob uporabniku v bazo zapišemo njegov `affiliate_id`, v primeru, da obstaja piškot, ki se je nastavil, ko je uporabnika na spletno stran napotil nek promotor.

4.3 Test na produkcijskem okolju

Za dokončno testiranje v produkcijskem okolju nam ne ostane nič drugega, kot da se na strani registriramo kot nov promotor in prek naše promocijske povezave opravimo registracijo ter nakup izdelka ter preverimo, če se vsi dogodki pravilno izvedejo in zapišejo v partnerskem sistemu.

Poleg tega moramo biti pozorni tudi na obnavljajoče naročnine, pri katerih moramo paziti, da se rezultat oziroma obračun ustrezno zabeleži vsakič, ko se nekemu uporabniku avtomatično obračuna storitev. V našem primeru moramo ob vsaki obnovi naročnine klicati API metodo `track_charge`, ki se navadno zgodi v mesečnem intervalu, ko prejmemo IPN [43] obvestilo od PayPal.

4.4 Primer uporabe

Marjan se ukvarja s spletnim marketingom že nekaj časa. Storitev RankTrackr mu je všeč, saj jo uporablja za svoje strani in tudi svoje stranke, hkrati pa ve, da bi lahko aplikacijo priporočal na svojem blogu ali svojim znancem in za to dobil primerno nagrado.

Marjan se registrira v našem partnerskem sistemu. Naš sistem mu priredi povezavo, katero lahko uporabi, da z njo napoti svoje stranke ali znance na predstavitveno stran aplikacije RankTrackr. Ko določen uporabnik obvišče stran preko te povezave, ki vsebuje enolično določen parameter, ki označuje Marjana kot napotitelja, se na uporabnikov računalnik namesti piškot, ki hrani podatek o tem, kdo je tega uporabnika napotil na našo stran.

Marjan storitev predlaga Mateju, tako da mu razloži uporabnost storitve in mu pošlje svojo povezavo za promoviranje izdelka. Matej se ne odloči

za nakup takoj, vendar se vpiše na poštne novice in se na stran vrne čez nekaj dni. Tokrat se odloči za registracijo, in kasneje za nakup. Naš sistem ob registraciji preveri prisotnost piškota, ki opisuje, kdo je tega uporabnika napotil na našo storitev. V tem primeru ugotovi, da je to Marjan, in ta podatek ob registraciji zabeleži. Hkrati na naš partnerski sistem prek API vmesnika sporoči registracijo in plačilo, pri čemer zaračuna provizijo, ki jo Marjan pri tem dobi in ga o tem obvesti.

Poglavje 5

Zaključek

V sklopu diplomske naloge smo na temelju modernih razvojnih metodologij razvili spletno aplikacijo, ki nam omogoča prodajo in upravljanje partnerskih programov naših lastnih produktov in aplikacij. V precej kratkem času se nam je pridružilo kar nekaj partnerjev, ki nam na mesečni bazi naslavljajo nove stranke in pomagajo pri prodaji izdelkov. S principi in filozofijo modernih razvojih metodologij nam je uspelo sistem razviti v razmeroma kratkem času in učinkovito prilagoditi glede na potrebe in omejitve tehnologij. Rezultat takšnega pristopa je izdelek, ki je v zelo kratkem času dosegel svoja pričakovanja in daje zelo dobro osnovo za nadaljni razvoj in prilagoditve glede na zahtevnejše potrebe na trgu.

Ob relativno majhnem vložki dela nam je uspelo razviti fleksibilni sistem, ki z majhnimi posegi omogoča, da lahko storitev ali izdelek začnemo tržiti prek tržnih partnerjev. Obenem za tovrstno funkcionalnost nimamo nobenih dodatnih stroškov, razen stroške strežnikov, ki so minimalni. Poleg tega bi lahko v prihodnosti storitev začeli prodajati še drugim, ki jo potrebujejo.

5.1 Napake in popravki

Pri razvoju in testiranju na projektu smo naleteli tudi na nekaj napak.

Dve izraziti napaki, ki smo ju ugotovili šele na produkcijskem strežniku:

tracker_controller.rb

```
1  class TrackerController < ApplicationController
2    before_action :permit_params
3    before_action :set_product
4    skip_before_action :verify_authenticity_token #, if: :json_request?
5    after_action :set_access_control_headers
6
7    def show
```

Slika 5.1: Prilagoditve kontrolerja za Javascript vtičnik

Zapisovanje piškota

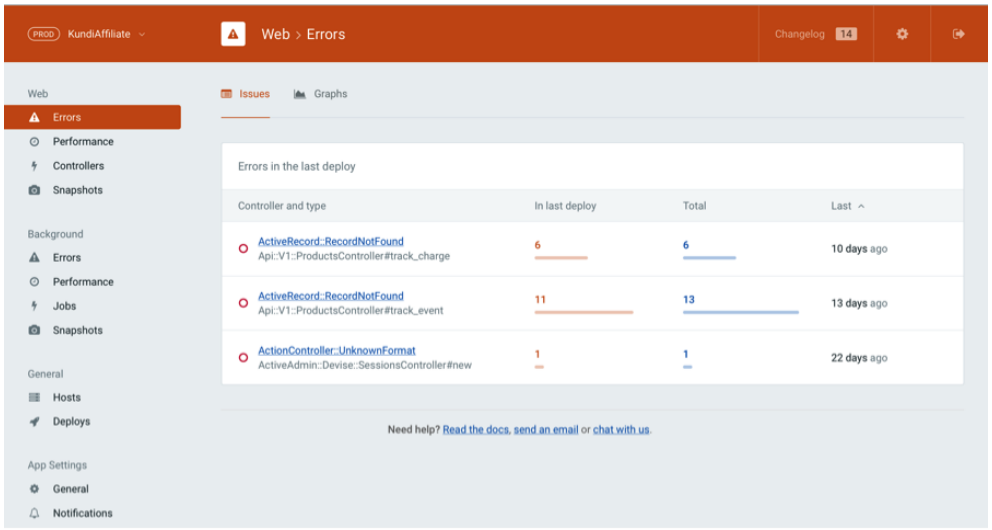
Zapisovanje piškota na odjemalcu se ni izvedlo na celotnem domenskem prostoru (.domain.com), ampak samo za aktivno domeno oziroma poddomeno [44]. Posledica tega je bila, da se statistike niso ustrezno beležile, ko je uporabnik preklopil med poddomenami, saj se je podatek o njegovem promotorju (napotitelju) skril oziroma ni bil dosegljiv.

Klici API funkcij iz Javascript vtičnika

Težave so bile tudi s klicem API funkcij prek Javascript vtičnika, kjer uporabljamo Json oz Jsonp, da lahko sistemu pošljemo podatek o obisku strani. Težava je bila v tem, da Ruby on Rails po privzeti nastavitvi zahteva preverjanje avtentičnosti zahtevka, kar pa v primeru tega klica ne moremo zagotoviti. Zato je bilo potrebno za tovrstno obnašanje aplikacijo opremiti s povratno metodo in prilagojeno glavo odziva (slika 5.1) [45].

K sreči nam pri spremljanju napak pomaga AppSignal aplikacija (slika 5.2), ki nas o vsaki napaki obvesti po e-pošti, hkrati pa nudi seznam in detajle vseh napak, ki so se zgodile po zadnji uvedbi kode na strežnik.

Pogoste so tudi napake, ki jih ne moremo odpraviti, saj na njih nimamo vpliva (so odvisne od ostalih uporabnikov oziroma napačne uporabe sistema) – na primer: klic API funkcije s ključem, ki ne obstaja; dostop do zapisa, ki ga ni v bazi. Kljub temu da teh napak ne moremo odpraviti, je dobro, da smo o vseh teh napakah obveščeni in jih aktivino spremljamo, saj le tako lahko vidimo, če je z aplikacijo vse v redu.



Slika 5.2: Aplikacija AppSignals

5.2 Nadaljni razvoj

V tem poglavju bomo opredelili možne izboljšave sistema in tržno strategijo, ki bi bila učinkovita pri prodaji rešitve.

5.2.1 Tehnične izboljšave

Tehničnih izboljšav na projektu je precej, zato se bomo osredotočili na najpomembnejše funkcionalnosti, za katere menimo, da bi rešitev postavile daleč pred ostale obstoječe rešitve.

Direktna integracija s PayPal

V sistem bi bilo dobro neposredno integrirati PayPal tako, da bi lahko provizije direktno izplačevali našim promotorjem, ne da moramo izvažati podatke o provizijah in jih delno ročno procesirati. Sistem bi tako poskrbel za avtomatizacijo procesa, seveda pa bi morali dodati nekakšne varovalke, da bi to opravilo moral odobriti administrator oziroma ga časovno omejiti, da se lahko zgodi šele po roku, ko uporabniki od PayPal ne morejo več zahtevati

povračila plačila.

Zaščita pred zlorabo

Uporabniki trenutno lahko na različne načine zlorabijo sistem (kar je pomanjkljivost tudi pri drugih sistemih). To je možno narediti tako, da se registrirajo prek računa prijatelja, ustvarijo svoj partnerski račun in se sami vpišejo prek partnerske povezave. V tem primeru pridobijo povračilo dela stroškov za uporabniški račun.

Za zaščito pred tovrstnimi scenariji imamo trenutno pogoj, da mora promotor pridobiti minimalno 2 različni stranki, preden lahko zahteva izplačilo. Ta pogoj do neke mere pomaga, vendar pa nikakor ne v absolutnem smislu.

Kljub temu da popolna zaščita za tovrstne scenarije ne obstaja, bi lahko dodali napredno spremljanje IP naslovov in brskalnikov uporabnikov na zaznanih asociacijah med njihovimi računi.

Podpora za več-nivojske partnerske programe

Kompleksnejši partnerski sistemi omogočajo več-nivojske partnerske programe, kar pomeni, da lahko promotor pod sebe dobi več promotorjev in tako od vsakega prejme del provizij. Tovrstni sistemi so primerni za kompleksnejše promocije in promocijske mreže. Ta funkcionalnost pa bi omogočila tudi pridobitev pozornosti nekaterih drugih uporabnikov takšnih rešitev.

Napredne zmožnosti zaračunavanja provizij

V sistem bi bilo dobro vključiti tudi naprednejše zmožnosti zaračunavanja provizij. Trenutno omogočamo osnovne nastavitve zaračunavanja kot so: zaračunavanje na podlagi odstotka od celotnega zneska prodaje ali fiksnega zneska. Naprednejše možnosti bi omogočale nastavljanje zaporednih prodaj, na primer: pri prvi prodaji uporabnik prejme 10%, pri naslednji 12% itd. Sorodno bi lahko naredili tudi za zaporedje provizij fiksnih zneskov.

5.2.2 Tržna strategija

V tem delu bomo predstavili tržno strategijo, za katero menimo, da bi bila učinkovita pri prodaji partnerskega sistema.

Predstavitev

Na spletni strani kundiaffiliate.com bi predstavili glavne 3 prednosti našega partnerskega sistema:

- Enostavnost integracije.
- Podpora za Mixpanel.
- Ugodna cena.

Za začetek bi ponudili poceni osnovni paket, s fiksno ceno (brez provizij transakcij) 29 USD na mesec (konkurenčna storitev Ambassador ima najcenejši paket za mesečno plačilo 300 USD).

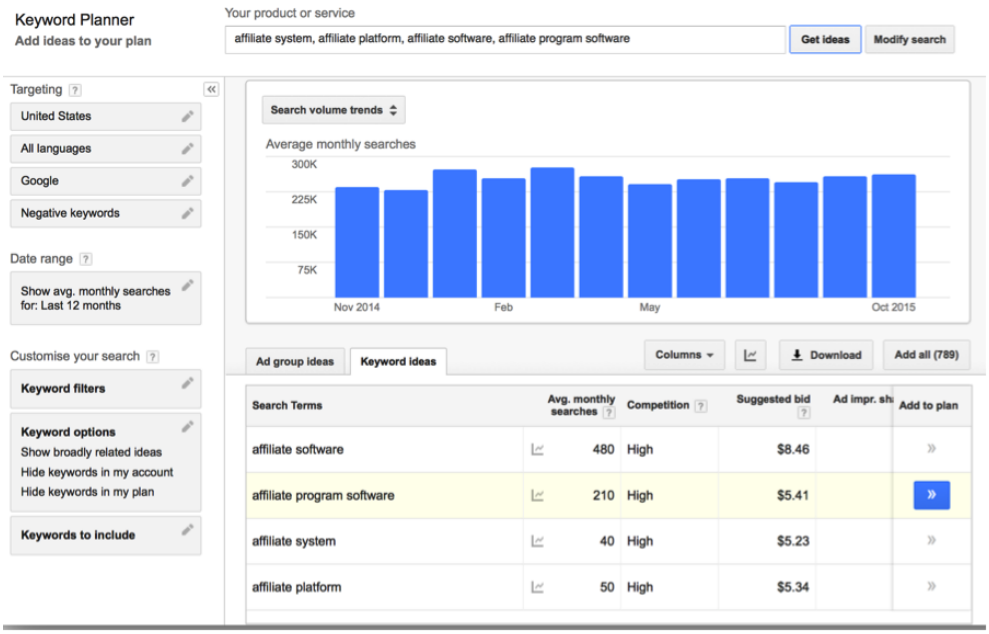
V nadaljevanju bi paket prilagodili, odvisno od tega, koliko bi uporabniki bili pripravljeni plačati za nadgrajeno rešitev in glede na izboljšave, ki bi jih dodali v sistem.

Del tržne strategije bi bila tudi video predstavitev, v kateri bi prikazali, kako enostavno je rešitev implementirati v obstoječi sistem. Ta predstavitev bi bila predvsem ciljana na razvijalce, saj se oni pogosto odločijo in prepričajo svoje vodje za primerno rešitev na tehnološkem področju.

Pridobitev interesentov

Za oglaševanje bi uporabil Google Adwords, Facebook oglase in sponzorirane PR objave na raznih relevantnih straneh kot so entrepreneur.com, getapp.com in ostalih sorodnih portalih.

Ciljna skupina uporabnikov bi bili lastniki spletnih strani, starejši od 20 let. Primarna ciljna skupina so lastniki SaaS storitev, ki že imajo ali pa še



Slika 5.3: Potencial trga z uporabo orodja Google Keyword Planner

nimajo partnerskih sistemov, sekundarna skupina pa lastniki spletnih trgovin ali prodajalci informativnih produktov (kot so na Clickbank, YVZoo). Osredotočil bi se na ameriški in angleški trg, v kasnejši fazi pa na Evropo.

Na Facebook bi lahko za kampanjo uporabili tudi Facebook stran sorodnih rešitev, prav tako bi lahko na Google iskalniku zakupili ključne besede, ko uporabniki iščejo splošno partnersko rešitev ali pa katero od že znanih oziroma obstoječih platform. Slika 5.3 prikazuje količino mesečnih iskanj oziroma potencial na iskalniku Google za našo tržno nišo z uporabo orodja Google Keyword Planner [48].

5.3 Učne lekcije

S projektom smo se naučili veliko praktičnih in pa tudi bolj abstraktnih stvari, ki nam bodo v prihodnosti pomagale razumeti proces in sprejemati odločitve za še boljšo učinkovitost. Te bi lahko strnili na sledeča spoznanja:

5.3.1 Izkoriščanje naleta motivacije

Pomembna stvari, ki smo se je naučili pri razvoju projekta je, da je dobro biti dovzeten za prebliske motivacije in te maksimalno izkoristiti, ko se zgodijo. Te bi lahko definirali kot krajša obdobja višjega energijskega potenciala in motivacije, ki navadno trajajo od 2 do 3 dni in so optimalni čas za izkoriščanje razvoja kompleksnejših delov.

5.3.2 Strogozačrtan časovni plan

Pri delu na projektu smo se naučili, da je v smislu časovnih okvirov zelo dobro imeti natančno in strogo začrtan časovni plan v smislu željenega cilja. To nas prisili, da se oddaljimo od teženj po nepotrebnih optimizacijah in abstrakciji kode, kadar to v resnici ni potrebno.

5.3.3 Selekcija obstoječih komponent

Pri uporabi že napisanih delov kode ali komponent je pomembno, da se zavedamo tega, da je dobro, da uporabimo čimveč kar se da, vendar pa moramo pri tem paziti na nekaj stvari. Prva je ta, da je projekt, ki ga vključimo oziroma uporabimo v našem projektu ustrezno vzdrževan oziroma ni zastarel. Druga stvar je ta, da za enostavnejše funkcionalnosti ne težimo vedno k uporabi knjižic, saj se moramo zavedati, da nam vsaka komponenta doda eno odvisnost več k projektu, s tem pa raste tudi kompleksnost vzdrževanja.

5.3.4 Abstrakcija funkcionalnosti

Zelo pomembna stvar, ki smo se jo naučili z uporabo Scrum metodologije na projektu, je učenje o ravnovesju programiranja med abstrakcijo in specifičnostjo. Problem, ki se nam zgodi, če razmišljamo o razvoju programske opreme preveč vnaprej je ts, da se lahko ujamemo v pretirani abstrakciji kode, za kar porabimo preveč časa. Moderne metodologije pa nas silijo, da se držimo rokov in ciljev, in se tako temu izognemo.

Literatura

[1] (2016) Wikipedia, Affiliate Marketing. Dosegljivo: https://en.wikipedia.org/wiki/Affiliate_marketing.

[2] (2016) Study.com, Cross Promotion Definition. Dosegljivo: http://study.com/academy/lesson/cross-promotion-definition-ideas-examples.html.

[3] (2016) Wikipedia, Self-hosting. Dosegljivo: https://en.wikipedia.org/wiki/Self-hosting.

[4] (2016) SearchItChannel, Managed-hosting. Dosegljivo: http://searchitchannel.techtarget.com/definition/managed-hosting.

[5] (2016) Mixpanel. Dosegljivo: http://mixpanel.com.

[6] (2016) iDevAffiliate. Dosegljivo: http://idevdirect.com.

[7] (2016) PostAffiliatePro. Dosegljivo: https://www.postaffiliatepro.com/.

[8] (2016) Ambassador. Dosegljivo: https://www.getambassador.com/.

[9] (2016) Wikipedia, Agile Software Development. Dosegljivo: https://en.wikipedia.org/wiki/Agile_software_development.

[10] (2016) Ruby on Rails. Dosegljivo:
http://rubyonrails.org/.

[11] (2016) Dictionary.com, Metodology. Dosegljivo:
http://www.dictionary.com/browse/methodology.

[12] (2016) Techrepublic.com, Heavyweight vs. lightweight methodologies. Dosegljivo:
http://www.techrepublic.com/article/heavyweight-vs-lightweight-methodologies-key-strategies-for-development/.

[13] Scott Belsky, *Making Ideas Happen: Overcoming the Obstacles Between Vision and Reality*, Portfolio, 2012.

[14] (2016) Agile Manifesto. Dosegljivo:
http://agilemanifesto.org/.

[15] Ken Schwaber, *Agile Software Development with Scrum*, Pearson, 2001.

[16] (2016) Pivotal Tracker. Dosegljivo:
http://pivotaltracker.com.

[17] (2016) Youtube, Test-Driven Development. Dosegljivo:
https://www.youtube.com/watch?v=dWayn0QsJr8.

[18] Jeff Sutherland, *Scrum: The Art of Doing Twice the Work in Half the Time*, Crown Business, 2014.

[19] (2016) MountainGoatSoftware, Agile User Stories. Dosegljivo:
https://www.mountaingoatsoftware.com/agile/user-stories.

[20] (2016) nGinx. Dosegljivo:
http://nginx.org/.

[21] (2016) Puma, Gem. Dosegljivo:
https://github.com/puma/puma.

[22] (2016) TeamTreeHouse, Frontend vs. Backend. Dosegljivo: `http://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend`.

[23] (2016) AppSignal. Dosegljivo: `https://appsignal.com/`.

[24] (2016) Devise, Gem. Dosegljivo: `https://github.com/plataformatec/devise`.

[25] (2016) ActiveAdmin, Gem. Dosegljivo: `https://github.com/activeadmin/activeadmin`.

[26] (2016) Varonis Blog, Introduction to oAuth. Dosegljivo: `https://blog.varonis.com/introduction-to-oauth/`.

[27] (2016) Postgresql. Dosegljivo: `http://www.postgresql.org/`.

[28] (2016) Postgresql, hStore. Dosegljivo: `http://www.postgresql.org/docs/9.0/static/hstore.html`.

[29] Michael Hartl, *Ruby on Rails Tutorial: Learn Web Development with Rails (3rd Edition)*, Addison-Wesley, 2012.

[30] (2016) Zurb, Foundation: Dosegljivo: `http://foundation.zurb.com/`.

[31] (2016) What is REST. Dosegljivo: `http://rest.elkstein.org/2008/02/what-is-rest.html`.

[32] (2016) jQuery. Dosegljivo: `https://jquery.com/`.

[33] (2016) StackOverflow, jQuery isolation. Dosegljivo: `http://stackoverflow.com/questions/14717764/jquery-isolation`.

[34] (2016) Redis. Dosegljivo:
http://redis.io/.

[35] (2016) Sidekiq, Gem. Dosegljivo:
https://github.com/mperham/sidekiq.

[36] (2016) rBenv. Dosegljivo:
https://github.com/rbenv/rbenv.

[37] (2016) Ruby on Rails, Active Support Concern. Dosegljivo:
http://api.rubyonrails.org/classes/ActiveSupport/Concern.html.

[38] (2016) High Charts. Dosegljivo:
http://www.highcharts.com/.

[39] (2016) RankTrackr. Dosegljivo:
http://ranktrackr.com.

[40] (2016) Capistrano, Gem. Dosegljivo:
https://github.com/capistrano/capistrano.

[41] (2016) DigitalOcean. Dosegljivo:
https://www.digitalocean.com/.

[42] (2016) Git Revision Selection. Dosegljivo:
https://git-scm.com/book/en/v2/Git-Tools-Revision-Selection.

[43] (2016) Paypal, Instant Payment Notification. Dosegljivo:
https://developer.paypal.com/webapps/developer/docs/classic/products/instant-payment-notification/.

[44] (2016) Erik.io, Guide to Cookie Domains. Dosegljivo:
http://erik.io/blog/2014/03/04/definitive-guide-to-cookie-domains/.

[45] (2016) StackOverflow, Understanding Rails Authenticity Token. Dosegljivo:
`http://stackoverflow.com/questions/941594/understanding-the-rails-authenticity-token`.

[46] (2016) Agile Metodology. Dosegljivo:
`http://agilemethodology.org/`.

[47] Steve Blank, *The Startup Owner's Manual*, K & S Ranch, 2012.

[48] (2016) Google Keyword Planner. Dosegljivo:
`https://adwords.google.com/KeywordPlanner`.